Ministério da Agricultura,
Pecuária e Abastecimento

Boletim de Pesquisa e Desenvolvimento 4

Outubro, 2002

ISSN 1677-9266

Teste de Fluxo de Dados de Programas com Ponteiros e Registros

Embrapa

Embrapa

ISSN 1677-9266
Outubro, 2002

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Informática Agropecuária
Ministério da Agricultura, Pecuária e Abastecimento

Boletim de Pesquisa e Desenvolvimento 4

Teste de Fluxo de Dados de Programas com Ponteiros e Registros

Marcos Lordello Chaim
José Carlos Maldonado
Mario Jino

Campinas, SP
2002

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Av. André Tosello, 209
Cidade Universitária "Zeferino Vaz" – Barão Geraldo
Caixa Postal 6041
13083-970 – Campinas, SP
Telefone (19) 3789-5743 - Fax (19) 3289-9594
URL: http://www.cnptia.embrapa.br
e-mail: sac@cnptia.embrapa.br

Supervisor editorial**:** *Ivanilde Dispato*
Normalização bibliográfica: *Marcia Izabel Fugisawa Souza*
Capa: *Intermídia Produções Gráficas*
Editoração eletrônica: *Intermídia Produções Gráficas*

1ª. edição
on-line - 2002

Chaim, Marcos Lordello.
Teste de fluxo de dados de programas com ponteiros e registros / Marcos Lordello Chaim, José Carlos Maldonado, Mario Jino. — Campinas : Embrapa Informática Agropecuária, 2002.
35 p. : il. — (Boletim de Pesquisa e Desenvolvimento / Embrapa Informática Agropecuária ; 4)

ISSN 1677-9266

1. Teste de software. 2. Teste de fluxo de dados. 3. Análise de fluxo de dados. I. Maldonado, José Carlos. II. Jino, Mario. III. Título. IV. Série.

CDD – 005.14 (21st ed.)

Sumário

Teste de Fluxo de Dados de Programas com Ponteiros e Registros

Marcos Lordello Chaim[1]
José Carlos Maldonado[2]
Mario Jino[3]

Resumo

Dois modelos mais precisos de análise de fluxo de dados voltados para o teste de programas que utilizam ponteiros e campos de registros são apresentados. Os modelos propostos são baseados em uma abordagem conservadora e foram implementados na ferramenta POKE-TOOL. A conjectura é que a análise de fluxo de dados mais precisa aumenta a eficácia do teste a um custo razoável. Para investigar essa conjectura, um estudo de caso foi realizado para avaliar a eficácia e o custo da utilização dos dois modelos no teste com duas famílias diferentes de critérios de teste baseados em análise de fluxo de dados. Os resultados obtidos indicam que a eficácia dos modelos dependem do *programa* e dos *defeitos* que ele contém. A segunda observação em especial indica que existem defeitos cuja detecção é facilitada quando os modelos propostos são utilizados. Para uma das famílias de critérios de teste, foi observado que o custo adicional causado pela análise mais precisa de fluxo de dados é razoável.

Termos de indexação: teste baseado em análise de fluxo de dados, análise de fluxo de dados precisa, eficácia, custo.

---

[1] Mestre e Doutor em Engenharia Elétrica, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. Durante a realização deste trabalho, foi apoiado parcialmente pela Fundação CAPES.

[2] Mestre em Sistemas de Computação e Doutor em Engenharia Elétrica, Prof. do Instituto de Ciências Matemáticas e de Computação, USP, Caixa Postal 668 – 13560-970 – São Carlos, SP.

[3] Mestre em Engenharia Elétrica e Doutor em Ciências da Computação, Prof. da FEEC/Unicamp, Caixa Postal 6101 – 13083-970 – Campinas, SP.

Data Flow Testing of Programs with Pointers and Records

Abstract

Two data-flow models for implementation of accurate coverage assessment of pointers and records in data-flow testing are described. The models are based in a conservative approach and were implemented in POKE-TOOL (Potential Uses Criterial Tool for Program Testing). The conjecture is that accurate data-flow analysis would improve the effectiveness of data-flow testing criteria at a reasonable cost. To assess its cost and effectiveness, we evaluated the adequacy of tests with respect to two different data-flow testing criteria families implemented using the conservative data-flow models. The effectiveness and size of these adequate test sets were compared. Results of the comparison indicate that the effectiveness of the conservative data-flow models depends on the program and the fault(s) it contains. Moreover, there exist faults whose detection is facilitated by a more accurate data-flow analysis such as that proposed by the conservative data-flow models. For one of the testing criteria investigated, we also found that the extra cost of analysis and coverage measurement for pointers and records is reasonable.

Index terms: data-flow testing; accurate data-flow analysis, effectiveness; and cost.

Introdução

Vários critérios de teste (Ntafos, 1984; Rapps & Weyuker, 1985; Ural & Yang, 1988; Maldonado et al., 1992a; Herman, 1976; Laski & Korel, 1983) e ferramentas de apoio a sua aplicação (Horgan & London, 1991; Chaim, 1991; Frankl et al., 1985; Agrawal et al., 1998; Ostrand & Weyuker, 1991; Marx & Frankl, 1999) foram desenvolvidos durante os anos 1980s e 1990s visando criar conjuntos de teste mais efetivos, isto é, que revelam a presença de defeitos. Hoje esses critérios e ferramentas definem o estado da arte em técnicas de teste e ferramentas. Os critérios baseados em análise de fluxo de dados estabelecem requisitos de teste que exigem a execução de caminhos entre a *definição* (atribuição de valor) e o *uso* (subseqüente referência) de uma variável; por isso, os seus requisitos de teste são chamados de *associações de fluxo de dados*. A motivação desses critérios é que ao *exercitar* as associações de fluxo de dados o testador tem aumentada a sua confiança em relação à corretitude do programa.

As definições dos critérios de teste baseados em fluxo de dados são abstratas e não estabelecem como os conceitos de fluxos de dados básicos (e.g., *definição*, *uso* e *associação de fluxo de dados*) são implementados para uma linguagem em particular. Porém, a aplicação desses critérios em programas reais requer o uso de um *modelo de dados* que define como esses conceitos são tratados com respeito aos recursos mais comuns das linguagens de programação, tais como, variáveis agregadas (vetores e matrizes), variáveis estruturadas (registros) e variáveis derreferenciadas (endereços de memória obtidos pela derreferenciação de ponteiros) (Ghezzi & Jazayery, 1987). O desenvolvedor de ferramentas deve, portanto, selecionar um ou mais modelos de dados para apoiar o teste na linguagem alvo da sua ferramenta. Essa decisão tem implicações importantes não somente no custo e na eficácia dos critérios de fluxo de dados (como será discutido a seguir), mas também na posição relativa desses critérios na hierarquia de critérios de teste estruturais (Horgan & London, 1991; Frankl & Weyuker, 1988; Vergilio, 1997).

As primeiras versões das ferramentas de teste de fluxo de dados não consideravam os fluxos que envolviam a derreferenciação de ponteiros e o uso de campos de registros (Silva, 1995; Frankl et al., 1985; Chaim, 1991). No entanto, os defeitos devido ao uso incorreto desses recursos são comuns. Duncan & Robson (1996) desenvolveram um estudo exploratório dos defeitos mais comuns em programas escritos em linguagem C no qual mais de 70% dos programadores entrevistados responderam que os defeitos relativos ao

uso incorreto de ponteiros estavam entre os mais freqüentes. Portanto, abordagens de teste de fluxo de dados que tenham como alvo a detecção e localização desses defeitos são necessárias e desejáveis.

Este problema tem sido abordado na literatura aumentando-se a precisão da análise de fluxo de dados (Horgan & London, 1991; Vilela et al., 1997; Marx & Frankl, 1999, 1996; Ostrand & Weyuker, 1991). A conjectura é que a análise de fluxo de dados mais precisa aumenta a eficácia do teste a um custo razoável (Marx & Frankl, 1999, 1996). O nível de precisão, por sua vez, é definido pelo modelo de dados implementado pela ferramenta de teste.

Algumas ferramentas utilizam modelos de dados que não distinguem as variáveis derreferenciadas das variáveis agregadas (Horgan & London, 1991). Essas ferramentas consideram todos os endereços de memória possíveis de serem associados a uma variável derreferenciada como um único *objeto de memória*. Como conseqüência, uma atribuição (ou referência) a qualquer elemento do objeto de memória significa uma definição (uso) de todo objeto. A vantagem dessa abordagem é a sua simplicidade visto que requer apenas o registro do caminho executado para a análise de adequação dos casos de teste. Por outro lado, ela pode produzir uma medida de cobertura *inflacionada*, pois não há garantia de que os fluxos de dados das associações *consideradas* como exercitadas tenham realmente ocorrido em tempo de execução ou de que aqueles que realmente ocorreram foram identificados por associações *exercitadas*.

Outra maneira de aumentar a precisão do modelo de dados é pelo monitoramento das posições de memória. Ostrand & Weyuker (1991) consideram associações de fluxo de dados de *posições de memória*, de tal forma que somente existe uma associação de fluxo de dados se a mesma posição de memória recebe um valor e este valor, inalterado, é subseqüentemente usado. A vantagem dessa abordagem é que ela captura *precisamente* os fluxos de dados estabelecidos nas associações definição-uso. O efeito colateral é o custo adicional que ela acarreta devido à necessidade de monitoramento das posições de memória em tempo de execução.

Marx & Frankl (1999, 1996) estenderam o conceito de associação definição-uso para tratar ponteiros no teste de fluxo de dados. As associações definição-uso *estendidas* demandam a execução de caminhos livres de definição nos quais a posição de memória amarrada a uma variável derreferenciada não é alterada. De maneira semelhante à abordagem dinâmica de Ostrand & Weyuker (1991), a abordagem rastreia *precisamente* as cadeias definição-

uso das variáveis derreferenciadas, porém sem monitorar as posições de memória. Entretanto, esta solução tem implicações em termos de custo visto que o algoritmo para determinar as associações estendidas possui complexidade assintótica *super-exponencial* e pode requerer associações adicionais (Marx & Frankl, 1999, 1996). Além disso, ele não é adequado para linguagens que permitem operações aritméticas com ponteiros, o que inviabiliza a sua utilização nas linguagens que mais usam esse recurso (e.g., C, C++).

Nesse trabalho, são apresentados dois modelos mais precisos de análise de fluxo de dados voltados para o teste de programas que utilizam ponteiros e campos de registros. Os modelos propostos, chamados de **nível 1** e **nível 2**, implementam níveis incrementalmente maiores de precisão na análise de fluxo de dados e são baseados na abordagem conservadora (Vilela et al., 1997; Maldonado, 1991). Esta abordagem trata as posições de memória que *podem* ser endereçadas pelas variáveis derreferenciadas como um objeto comum de memória; porém, quando comparada com outras abordagens, ela é mais conservadora na determinação dos requisitos de teste.

Para verificar a conjectura de que a eficácia aumenta a um custo razoável, foi realizado um estudo de caso. Nesse estudo, a eficácia e o custo do teste utilizando os modelos **nível 1** e **nível 2** foram comparados aos do teste utilizando um modelo (**nível 0**) que não leva em consideração as definições e usos relativos à derreferenciação de ponteiros ou aos campos de registros. Os três modelos foram implementados na ferramenta POKE-TOOL e a *eficácia* e o *custo* foram analisados com respeito a várias versões *defeituosas* do programa `Sort` do sistema Unix.

O restante desse trabalho é organizado da seguinte forma. Os conceitos básicos de fluxo de controle e de dados, os critérios de teste utilizados neste Boletim de Pesquisa e os modelos de dados baseados na abordagem conservadora são definidos e apresentados em Material e Métodos. A descrição do estudo de caso, os resultados obtidos e sua análise estão contidos em Resultados e Discussão seguido das conclusões obtidas. A notação e o jargão utilizados são relativos à linguagem C.

Material e Métodos

Seja *P* um programa escrito em uma linguagem do estilo Algol (Ghezzi & Jazayery, 1987) e $S_1 ... S_i ... S_n$, $1 \leq i \leq n$, a sua seqüência de comandos. *P* pode ser mapeado para um grafo de fluxo de controle *G(N, R, s, e)* onde *N* é

o conjunto de blocos de comandos (*nós*) tal que uma vez executado o primeiro comando do bloco todos são executados seqüencialmente, *e* é o nó de entrada, *s* é o nó de saída e *R* é o conjunto de ramos que representam a possível transferência de fluxo de controle entre dois nós. A Fig. 2 apresenta o grafo de fluxo de controle obtido a partir do programa da Fig. 1. Um caminho é a seqüência de nós (n_i, ... , n_k, n_{k+1}, ... , n_j), onde $i \leq k < j$, tal que (n_k, n_{k+1}) $\in R$. Um caminho é livre de laço se todos os nós são distintos. Um caminho é completo quando começa em *e* e termina em *s*.

Uma *definição de variável* (*def*) ocorre sempre que um valor é armazenado em uma posição de memória. A ocorrência de uma variável é um *uso* quando ela não estiver sendo definida. Dois tipos de usos são distinguidos: *c-uso* e *p-uso*. O primeiro tipo afeta diretamente uma computação sendo realizada ou permite que o resultado de uma definição anterior possa ser observado. O segundo tipo afeta diretamente o fluxo de controle do programa.

Um caminho (i, n_1, ... , n_m, j), $m \geq 0$, que não contenha definição de uma variável x nos nós n_1, ... , n_m é chamado de *caminho livre de definição* com respeito a (c.r.a.) x do nó i ao nó j e do nó i ao arco (n_m, j). Um caminho livre de definição (n_1, n_2, ... , n_j, n_k) c.r.a. a uma variável x, onde o caminho (n_1, n_2, ... , n_j) é um caminho livre de laço e n_1 tem uma definição de x, é denominado potencial-du-caminho[4] c.r.a. x.

A intuição subjacente aos critérios baseados em análise de fluxo de dados (chamados simplesmente de critérios de fluxo de dados) (Ntafos, 1984; Rapps & Weyuker, 1985; Ural & Yang, 1988; Maldonado et al., 1992a; Herman, 1976; Laski & Korel, 1983) é que a confiança em relação à corretitude do programa é aumentada se todo valor atribuído a uma variável for utilizado pelo menos uma vez na execução do conjunto de casos de teste (Frankl & Weyuker, 1988). Os critérios Potenciais Usos (Maldonado et al., 1992a) estenderam essa intuição utilizando o conceito de *potencial uso*. Segundo este conceito, os requisitos de teste devem exigir caminhos entre uma definição e os pontos do programa onde o valor da definição *pode* ser utilizado, ou seja, onde há um potencial uso.

A seguir, são descritos alguns critérios de fluxo de dados das famílias Fluxo de Dados (Rapps & Weyuker, 1985) e Potenciais Usos (Maldonado et al., 1992a):

---

[4] Potencial du-caminho é a forma simplificada para caminho definição-uso potencial. A diferença entre caminho livre de definição e potencial du-caminho é que no primeiro podem ocorrer laços; no segundo, não.

- **Critério Todos P-usos**: requer que todas as associações definição-uso (*adu*) do tipo (*i*, (*j,k*), *x*) do programa em teste sejam exercitadas pelos casos de teste de um conjunto *T*. Uma associação (*i*, (*j,k*), *x*) é exercitada quando pelo menos um caminho livre de definição c.r.a. *x* do nó *i* até o ramo (*j,k*) é executado por um caso de teste $t \in T$.
- **Critério Todos Usos**: requer que todas as associações definição-uso dos tipos (*i*, *j*, *x*) e (*i*, (*j,k*), *x*) sejam exercitadas pelos casos de teste de um conjunto *T*. Uma associação (*i*, *j, x*) ou (*i*, (*j,k*), *x*) é exercitada quando pelo menos um caminho livre de definição c.r.a. *x* do nó *i* até o nó *j* ou ramo (*j,k*) é executado por um caso de teste $t \in T$.
- **Critério Todos Potenciais-Usos**: requer que todas as associações definição-potencial-uso (*adpu*) dos tipos (*i, j, x*) e (*i*, (*j,k*), *x*) sejam exercitadas pelos casos de teste de um conjunto *T*. Note-se que para caracterizar uma associação definição-potencial-uso não é necessário um uso explícito de *x* em *j* ou (j,*k*), apenas que o nó *j* ou ramo (j,*k*) seja alcançável por um caminho livre de definição c.r.a. *x* a partir de *i*.
- **Critério Todos Potenciais-Usos/Du**: também requer que todas as associações definição-potencial-uso dos tipos (*i, j, x*) e (*i*, (*j,k*), *x*) sejam exercitadas pelos casos de teste de um conjunto *T*, porém, por potenciais du-caminhos.

Os critérios de teste estruturais, por serem baseados em caminhos, podem exigir que caminhos não-executáveis sejam exercitados. Isto ocorre porque a determinação dos requisitos de teste é realizada considerando-se apenas aspectos sintáticos do programa. Um caminho é *não-executável* se não existir um conjunto de valores de entrada, parâmetros e variáveis globais que provoquem a sua execução.

Com o objetivo de reduzir o número de *adpu*s requeridas e, dessa maneira, reduzir o custo da análise de adequação dos critérios Potenciais Usos, Maldonado et al. (1992b) estenderam o conceito de *ramos essenciais* (Chusho, 1987). Os ramos essenciais constituem um subconjunto dos ramos do programa cuja propriedade é garantir a execução de *todos* os ramos do programa quando são executados. As *adpu*s foram então redefinidas utilizando a seguinte notação: (*i, (j,k), D*)), onde (*j,k*) é um ramo essencial estendido e *D* é um conjunto de variáveis definidas no nó *i*. Para satisfazer a *adpu* (*i, (j,k), D*), os casos de teste devem executar caminhos que alcancem

(*j,k*) e que são livres de definição (potenciais du-caminhos) c.r.a. todas as variáveis $x \in D$. O exercício de todas as *adpu*s representadas utilizando ramos essenciais estendidos por casos de teste do conjunto *T* garante a adequação ao critério todos potenciais-usos (todos potenciais usos/du).

Várias ferramentas foram desenvolvidas para apoiar a utilização dos critérios definidos: as ferramentas ASSET (Frankl et al., 1985), PROTESTE+ (Silva, 1995) e APODOS (Marx & Frankl, 1999, 1996) apóiam a aplicação dos critérios de Rapps e Weyuker para a linguagem Pascal; ATAC (Horgan & London, 1991) e Tactic (Ostrand & Weyuker, 1991) são ferramentas que apóiam a aplicação dos critérios de Rapps e Weyuker para a linguagem C; e a ferramenta POKE-TOOL (Chaim et al., 1998) apóia a aplicação dos critérios todos nós, todos ramos e das famílias Fluxo de Dados e Potenciais Usos para várias linguagens de programação (C, FORTRAN, COBOL).

Os modelos de dados baseados na abordagem conservadora que mapeiam os conceitos e associações de fluxos de dados considerando as estruturas mais comuns das linguagens de programação (e.g., variáveis estruturadas, variáveis derreferenciadas, etc.) são descritos a seguir e apresentados também um exemplo de aplicação dos modelos propostos e uma análise dos custos envolvidos na aplicação dos modelos.

Vilela et†al. (1997) propuseram a utilização da *abordagem conservadora* no teste de fluxo de dados de programas com ponteiros e campos de registros. Esta abordagem foi inicialmente definida por Maldonado (1991) para tratar variáveis agregadas e parâmetros passados por referência na ferramenta POKE-TOOL. Ela parte do princípio de que é razoável superestimar o número de requisitos de teste para aumentar a sua eficácia.

Na análise dos fluxos de dados relativos a ponteiros e campos de registros, a abordagem conservadora é similar à de Horgan & London (1991) no sentido de que as variáveis derreferenciadas são tratadas da mesma forma que as variáveis agregadas. Porém, ela requer um número maior de requisitos de teste devido ao seu carácter conservador. Assim, segundo essa abordagem, qualquer atribuição (ou referência) a uma posição de memória endereçada utilizando expressões envolvendo ponteiros (e.g.,`*(p+1)` onde `p` é um ponteiro) aponta para o mesmo objeto comum de memória (e.g., `*p`). As definições de variáveis derreferenciadas dão origem a associações de fluxo de dados da mesma maneira que as definições de variáveis escalares, porém, diferentemente dessas últimas, não bloqueiam definições anteriores, o que implica um aumento no número de requisitos de teste exigidos pelo critério de fluxo de dados, como exemplificado a seguir.

Considere os comandos 34, 40 e 50 do programa exemplo 1 da Fig. 1. De acordo com a abordagem de Horgan & London (1991), não há distinção entre variáveis escalares, agregadas e derreferenciadas. Dessa maneira, a definição de `*p` em 34 não gera associações de fluxo de dados porque a definição de `*p` no comando 40 bloqueia a definição prévia no comando 34. Já de acordo com a abordagem conservadora a definição de `*p` no comando 40 não bloqueia a definição prévia em 34, resultando na exigência das associações (34,50,`*p`) e (40,50,`*p`), uma para cada atribuição a `*p`. A idéia é que as atribuições a uma variável derreferenciada não devem bloquear a exigência de outras associações de fluxo de dados (com respeito à mesma variável) porque em geral não é possível saber em tempo de compilação qual posição de memória está sendo alterada.

```
proc(condition1, condition2)
        int condition1, condition2;
{
                int x, y, z, *p;
24:                z = 17;
25:                x = 13;
26:                if(condition1) {
28:                        p=&y;
29:                        *p=z;
                   }
                   else {
33:                        p=&y;
34:                        *p=z;
35:                        switch(condition2) {
36:                              case 1:  p = &x;
37:                                       break;

38:                              case 2:  p = &z;
39:                                       break;
                         }
40:                      *p = 7 + z;
42:                      y = 53;
                }
49:                x = x + y + z;
50:                *p = *p +5;
50:                *p = *p +5;
51:                y = x + y;
```

Fig. 1. Programa exemplo 1.

A desvantagem da abordagem conservadora são as associações adicionais que ela requer devido aos seus requisitos de teste conservadores. Ela demanda mais associações do que a abordagem de Horgan & London (1991), o que implica um custo adicional para exercitá-as ou determinar sua *não-executabilidade*. Por outro lado, a abordagem conservadora minimiza a probabilidade de que fluxos de dados que ocorrem em tempo de execução deixem de ser exercitados pelo conjunto de casos de teste adequado visto que ela requer as associações (fluxos de dados) adicionais que *podem* ocorrer em tempo de execução. Outro aspecto importante é que ela não requer monitoramento de memória, exigindo tão somente o caminho executado pelo caso de teste para a análise de adequação.

O tratamento de ponteiros e campos de registros baseado na abordagem conservadora foi implementado na ferramenta POKE-TOOL. Três diferentes modelos de dados para tratamento de ponteiros e campos de registros estão disponíveis; eles são referenciados como **nível 0**, **nível 1** e **nível 2** porque fornecem níveis crescentes de precisão na análise de fluxo de dados.

O **nível 0** não leva em consideração associações de fluxo de dados envolvendo derreferenciação de ponteiros ou campos de registros. Este nível implementa o modelo de dados original da POKE-TOOL e utiliza a abordagem conservadora original para o tratamento de vetores e passagem de parâmetros por referência. Assim, de acordo com o **nível 0**, um vetor é tratado como uma variável simples. Por exemplo, há uma definição e uso de um vetor `v` no comando `v [ i] = 20 + v [ j] ;`, independentemente de qual elemento de `v` é realmente definido ou usado. Entretanto, a abordagem conservadora estabelece que as atribuições a elementos de vetores são *definições por referência* (*def-ref*). Este tipo de definição dá origem a novas associações, porém, os caminhos que incluem a *def-ref* continuam a ser livres de definição com relação a outras definições da mesma variável, ou seja, uma *def-ref* não bloqueia a geração de associações de fluxo de dados devido a definições prévias. Com relação às chamadas de procedimentos neste modelo, sempre que uma passagem por referência ocorre (e.g., `foo(&v)`), há uma *def-ref* e um uso do parâmetro real (e.g., `v`).

O **nível 1** implementa a abordagem conservadora para tratar os fluxos de dados relativos a derreferenciação de ponteiros e a campos de registros. De acordo com esse modelo, uma *variável derreferenciada* `*p` é definida por uma operação de derreferenciação sobre uma expressão envolvendo o ponteiro `p`. Por exemplo, as variáveis derreferenciadas `*p` e `**q` ocorrem, respectivamente, em expressões `*(p+i)` e `*q[ i+4]`. Outra diferença entre

os modelos **nível 1** e **nível 0** é que cada campo de uma variável estruturada (registro) é considerado como uma variável independente. Assim, se `r` é uma variável do tipo registro, cada campo `r.field` é uma variável diferente. De igual maneira, se a variável derreferenciada é do tipo registro (e.g., `*p`), então cada campo (e.g., `*p.field`) representa uma variável derreferenciada diferente. As variáveis derreferenciadas são tratadas da mesma forma que os vetores na abordagem conservadora. Portanto, uma atribuição a uma variável derreferenciada implica uma *def-ref*. Qualquer outra ocorrência é um uso.

Diferentemente de Horgan & London (1991), na abordagem conservadora, os resultados da aplicação do operador de derreferenciação uma vez (e.g., `*p`) ou duas vezes (e.g., `**p`) sobre um ponteiro (e.g., `p`) resultam em variáveis derreferenciadas distintas. A razão para esta distinção deve-se ao fato de que o tipo dos elementos `*p` é diferente do tipo dos elementos `**p`; logo, as variáveis derreferenciadas apontam para objetos de memória distintos. Além disso, a abordagem conservadora foi estendida para tratar as situações em que ponteiros e variáveis estruturadas (registros) são utilizados como parâmetros em chamadas de funções. Considere-se a função a seguir em que `v` é um ponteiro para um tipo `T`. O modelo de dados **nível 1** identifica as seguintes definições e usos:

```
foo(v, ...)
```

1. *c-uso* do ponteiro `v`;
2. *def-ref* de `*v`;
3. possível *c-uso* de `*v` (como foi considerada a existência de uma *def-ref* de `*v`, por uma questão de uniformidade, um *c-uso* de `*v` deve também ser considerado na chamada de procedimento);
4. se `T` é um registro, então todo campo de `*v` possui uma *def-ref* associada.

Se `v` é um ponteiro para ponteiro (e.g., `T ** v`), então o **nível 1** também estabelece a existência de uma *def-ref* de `**v` e um possível *c-uso* de `**v`.

O modelo de dados **nível 2** é uma extensão do **nível 1**. Ele trata as variáveis derreferenciadas e os campos de registros de maneira muito similar ao **nível 1**. A diferença está na forma como são tratadas as atribuições para os ponteiros. De acordo com o **nível 2**, quando um ponteiro `p` recebe um novo valor (i.e., é definido), a variável derreferenciada `*p` fica associada a uma nova posição de memória. Acontece que o valor de `*p` neste caso também é

novo, o que implica uma redefinição de `*p` também. Estas definições *implícitas* de variáveis derreferenciadas não são levadas em consideração no **nível 1**.

A intuição subjacente ao **nível 2** é que toda nova atribuição a um ponteiro deve ativar o teste da variável derreferenciada associada, uma vez que o valor atribuído ao ponteiro pode estar errado e o teste da variável derreferenciada pode aumentar as chances de detecção de um possível defeito. Este raciocínio já foi utilizado antes por Marx e Frankl (1999, 1996) e Pande et al. (1994).

A Tabela 1 descreve o número total de associações requeridas para o programa `Sort` (padrão do sistema Unix) para os critérios todos ramos, todos p-usos, todos usos, todos potenciais usos e todos potenciais usos/du para os diferentes níveis de análise fluxo de dados, bem como o aumento no número de associações provocado pela utilização dos níveis mais exigentes. Com relação a todos usos na Tabela 1, tem-se um total de 1.678 associações requeridas utilizando **nível 0**, 2.398 utilizando **nível 1** e 3.238 utilizando **nível 2**. Do **nível 0** ao **nível 1**, 720 associações extras são requeridas, causando um aumento de 42,91% no número de associações requeridas (valor indicado entre parênteses); do **nível 1** para o **nível 2**, 840 novas associações são requeridas (aumento de 35,03%); e do **nível 0** para o **nível 2**, 1.560 (aumento de 92,97%).

Tabela 1. Número de associações e o aumento causado pelos modelos de dados mais precisos.

Critério	*Modelo de dados*			*Aumento entre os modelos*		
	0	1	2	0-1	1-2	0-2
Todos Ramos	249	249	249	0	0	0
Todos P-usos	1071	1478	2063	407 (38,00%)	585 (39,58%)	992 (92,62%)
Todos Usos	1678	2398	3238	720 (42,91%)	840 (35,03%)	1560 (92,97%)
Todos Potenciais Usos“	4284	5263	5137	979 (22,85%)	-126 (-2,39%)	853 (19,91%)
Todos Potenciais Usos/Du	4284	5263	5137	979 (22,85%)	-126 (-2,39%)	853 (19,91%)

Tanto os critérios da família Fluxo de Dados como os da família Potenciais Usos são influenciados pelos modelos de dados. Entretanto, os critérios mais influenciados são todos p-usos e todos usos — por volta de 40% do **nível 0**

ao **nível 1**; e 90% do **nível 0** para o **nível 2**. Para os critérios todos potenciais usos, o aumento é de 22,85% do **nível 0** para o **nível 1** e de 19,91% do **nível 0** para o **nível 2**.

A razão pela qual os critérios Potenciais Usos são menos impactados pelos diferentes modelos de dados é devido à maneira como são determinadas as *adpu*s usando ramos essenciais estendidos. Como elas envolvem várias variáveis definidas em um nó, o uso dos **níveis 1** e **2** exigirá uma *adpu* adicional somente se uma definição de variável for detectada em um nó que não possuia uma variável definida de acordo com o **nível 0**.

Além disso, para os critérios Potenciais Usos, a introdução do **nível 2** levou a uma diminuição do número de associações em relação ao **nível 1**. Isto ocorre porque as definições implícitas de variáveis, que não são consideradas no **nível 1**, podem *bloquear* a existência de caminhos livres de definição que normalmente ocorrem com a utilização do **nível 1**. Este fenômeno é menos comum nos critérios da família Fluxo de Dados porque cada *adu* possui apenas uma variável, o que provoca um aumento maior no número de requisitos de teste pela simples introdução de novas definições de variáveis; porém, ele pode também ocorrer, dependendo da distribuição das definições implícitas consideradas no **nível 2**.

De qualquer maneira, o número de requisitos de teste exigidos pelos critérios de teste de fluxo de dados aumenta ao passar do **nível 0** para um modelo de dados que leva em consideração ponteiros e campos de registros (**nível 1** ou **nível 2**). O impacto desse aumento vai depender do modelo de implementação utilizado pelos critérios escolhidos.

Os custos envolvidos na utilização de modelos de dados mais precisos são analisados a seguir. Inicialmente, discutem-se os custos envolvidos no teste de fluxo de dados:

Geração de Casos de Teste

C_1 - o testador deve desenvolver um conjunto de casos de teste que seja adequado ao critério de fluxo de dados selecionado. Em geral, esse processo dá-se da seguinte maneira: um conjunto inicial de casos de teste é desenvolvido e a sua adequação ao critério é avaliada; se ele não é adequado ainda, então o conjunto deve ser aumentado com novos casos de teste até que se torne adequado.

Execução de Casos de Teste

C_2- os casos de teste devem ser executados para que sua validação possa ser realizada e as medidas de cobertura sejam coletadas.

Validação de Casos de Teste

C_3 - as saídas dos casos de teste devem ser validadas em relação à especificação do programa para a determinação dos casos de teste que revelam defeitos.

Análise de Adequação

C_4 - os dados coletados durante a execução dos casos de teste devem ser analisados para verificar a adequação do conjunto de casos de teste ao critério selecionado.

Análise de Não-executabilidade

C_5 - este custo inclui a verificação da não-executabilidade das associações de fluxo de dados. Trata-se de uma atividade muito custosa visto que essencialmente exige intervenção humana. Uma maneira de evitar a intervenção humana é pela utilização de uma solução aproximada. Ela consiste na geração de uma massa de casos de teste grande de tal forma que se um requisito de teste não é executado pela massa é assumido como *não-executável*. Esta solução é também cara porque pode exigir muitas horas de processamento para a geração, execução e validação dos casos de teste da massa de teste.

A utilização dos níveis mais precisos de análise de fluxo de dados tem implicações importantes nos custos C_1, C_2, C_3, C_4 e C_5. O aumento no número de associações devido aos **níveis 1** e **2** tende a aumentar o custo C_1 visto que associações adicionais devem requerer casos de teste extras. Dessa maneira, espera-se um aumento no tamanho dos conjuntos de casos de teste do **nível 0** para os demais. O aumento no tamanho dos conjuntos de casos de teste tem um impacto direto em C_2, C_3 e C_4: conjuntos maiores exigem a execução, validação e análise de adequação de mais casos de teste.

O custo C_4, por sua vez, é definido por duas variáveis: *o número de nós* do caminho executado pelo caso de teste e o *número de associações*. O algoritmo de análise de adequação implementado em algumas ferramentas (e.g., ASSET, POKE-TOOL) é baseado em autômatos finitos de tal forma que cada *associação* é vinculada a um *autômato finito* (chamados de descritores). Basicamente, o algoritmo testa todos os autômatos para possíveis transições sempre que um novo nó é visitado. Uma associação é considerada exercitada se o seu autômato atinge o estado final. Portanto, o custo da análise de adequação é $O(c \times a)$ onde c é o número de nós do caminho executado e a o número de autômatos. Como o tamanho do caminho executado permanece

constante, este custo pode ser reescrito como *O(a*. Assim, o aumento do número de associações (número de autômatos) *per se* aumenta o custo C_4.

A análise de não-executabilidade envolve a identificação de padrões de não-executabilidade (caminhos do programa para os quais não existem dados de entrada que provoquem a sua execução) que são comparados com os possíveis caminhos que exercitariam uma *adu* ou *adpu*. A tarefa mais difícil é a identificação desses padrões. O uso de um modelo de dados mais exigente pode implicar a necessidade de identificação de novos padrões de não-executabilidade visto que as associações adicionais tendem a requerer que novos caminhos sejam executados. Dessa maneira, C_5 deve aumentar com o crescimento do número de associações.

Da análise acima pode-se concluir que o *custo* de aplicação de um critério de fluxo de dados é principalmente determinado pelo *número de associações* e pelo *tamanho* do conjunto adequado de casos de teste. O aumento no número de *adu*s e *adpu*s causados pelo **nível 1** e **nível 2** certamente impacta os custos C_4 e C_5. Além disso, os modelos de dados que requerem mais associações tendem a gerar conjuntos de casos de teste maiores. Conjuntos maiores aumentam os custos C_1, C_2, C_3 e C_4. No experimento descrito em Resultados e Discussão, a *eficácia*, o *número de associações de fluxo de dados* e o *tamanho do conjunto de casos de teste* para diferentes critérios utilizando os três modelos de dados são analisados.

Resultados e Discussão

Neste experimento, o programa `Sort`, padrão do ambiente Unix, foi utilizado. O `Sort` possui várias vantagens para o tipo de comparação realizada: ele faz uso exaustivo de ponteiros e campos de registros (todas as funções usam variáveis do tipo ponteiro); e, embora seja um programa pequeno, algumas das suas funções são bastante complexas.

Wong (1993) desenvolveu uma massa de teste de 997 entradas geradas aleatoriamente para o teste de 25 versões defeituosas do programa `Sort`. Delamaro et al. (2001) analisaram a dificuldade de detecção dos defeitos dessas versões determinando o *índice de dificuldade* de cada um deles. Este índice foi determinado da segunte forma: 500 dados de teste foram gerados aleatoriamente e a porcentagem desses dados que revelavam o defeito foi calculada. Este processo foi repetido 30 vezes de forma que o índice final de dificuldade foi definido pela média das 30 repetições. Das 11 versões

defeituosas mais difíceis de detectar, foram identificadas sete versões que possuiam defeitos relacionados com o uso de ponteiros e campos de registros. A Tabela 2 descreve os sete defeitos selecionados bem como o índice de dificuldade (ID) de cada uma delas.

Tabela 2. Defeitos relativos ao uso incorreto de ponteiros e campos de registros introduzidos no programa Sort.

Defeito	*Código original*	*Código incorreto*	*ID*
1	if(b= *—ipb - *—ipa)	if(b= *—ipb - —*ipa)	10,60
2	while((*dp++ = *cp++) != '\n');	while((*++dp = *cp++) != '\n');	13,77
3	if(*—ipa != '0')	if(—*ipa != '0')	11,53
4	*pb == '\n' ? -fields[0].rflg	*pb == '\n' ? -fields[0].nflg	23,23
5	p->ignore = dict+128;	p->ignore = dict+290;	7,17
7	if (pb < lb && *pb==tabchar)	if (pb < lb ‖*pb==tabchar)	4,63
6	while(...ip[-1]->l<0)	while(...ip[-1]->l<=0)	3,99

O procedimento experimental utilizado consistiu nos seguintes passos. Inicialmente, a análise estática de cada versão defeituosa do `Sort` foi realizada de acordo com os três modelos de dados. Em seguida, todos os 997 casos de teste pertencentes à massa de teste foram executados e a adequação de cada caso de teste com respeito aos critérios selecionados — todos p-usos, todos usos, todos potenciais usos e todos potenciais usos/du — utilizando os três modelos de dados foi obtida.

As *adu*s e *adpu*s que não foram executadas por nenhum dos casos de teste da massa de teste foram consideradas *não-executáveis*. Portanto, os conjuntos de casos de teste considerados *adequados* com respeito a um critério são aqueles que possuem a maior medida de cobertura possível de se obter com a massa de teste.

O experimento consistiu na seleção de vários conjuntos de casos de teste adequados. Para cada um deles, foram determinados a *eficácia* - definida como o número de casos de teste reveladores de defeito no conjunto adequado - e o *tamanho* - definido como o número total de elementos do conjunto adequado. Utilizou-se o número de casos de teste reveladores de defeito como sendo a medida de eficácia porque todos os critérios de fluxo de dados, utilizando qualquer modelo, derivavam conjuntos adequados que incluíam pelo menos um caso de teste revelador de defeito. O tamanho dos conjuntos adequados, juntamente com o número de associações requeridas, estabelecem o *custo* de utilização de um critério de teste com um dado modelo de dados.

Os conjuntos adequados *T* foram obtidos da seguinte forma: casos de teste *t* da massa de teste eram escolhidos aleatoriamente e inseridos no conjunto *T* desde que pelo menos um requisito de teste diferente fosse executado; este processo era encerrado quando a máxima cobertura possível era atingida. Cem diferentes conjuntos de casos de teste adequados *T* foram selecionados e tiveram a sua eficácia e tamanho computados para cada versão defeituosa do `Sort` com respeito a cada critério de teste e modelo de dados.

A Tabela 3 apresenta a eficácia e o tamanho dos conjuntos adequados para os critérios todos ramos, todos p-usos, todos usos, todos potenciais usos e todos potenciais usos/du utilizando os modelos de dados **nível 0**, **nível 1** e **nível 2**. Por exemplo, a segunda linha refere-se aos dados relativos ao modelo de dados **nível 0** para o critério todos p-usos e as colunas indicam a versão defeituosa do `Sort`. Tanto as linhas como as colunas são subdivididas. As subdivisões das colunas indicam os dados relativos à eficácia (EF) e ao tamanho dos conjuntos adequados (Tam); as subdivisões das linhas referem-se à média (med) e ao desvio padrão (des) desses valores. Assim, para a versão 1, o critério todos p-usos no **nível 0** obteve os seguintes resultados: a eficácia média de 11,5 casos de teste reveladores de defeito com desvio padrão 2,1; o tamanho médio dos conjuntos adequados selecionados foi 53,2 casos de teste com desvio padrão 4,8. O critério todos ramos não é dividido em modelos de dados visto que se trata de um critério baseado unicamente em fluxo de controle.

A Tabela 4 mostra a eficácia de conjuntos de casos de teste de vários tamanhos selecionados aleatoriamente. Por exemplo, considere-se a linha que indica os conjuntos de casos de teste aleatórios de tamanho 50; eles apresentam para o defeito 1 uma média de 5,98 de casos de teste reveladores de defeito e um desvio padrão de 2,29 para cem repetições.

Os resultados do experimento são também apresentados na forma de gráficos. Nas Fig. 2, 3, 4 e 5 são apresentados gráficos que relacionam a eficácia e o tamanho dos conjuntos adequados com as versões do programa `Sort`. Para observar o efeito do aumento do número de casos de teste na eficácia, os dados relativos a conjuntos de casos de teste gerados aleatoriamente foram incluídos nos gráficos. O tamanho desses conjuntos aleatórios compreendem a faixa de variação do tamanho dos conjuntos adequados utilizando os diferentes modelos de dados.

Tabela 3. Eficácia (EF) e tamanho (Tam) de conjuntos adequados a vários critérios de teste.

Critério	Modelo de dados		Defeitos 1 EF	1 Tam	2 EF	2 Tam	3 EF	3 Tam	4 EF	4 Tam	5 EF	5 Tam	6 EF	6 Tam	7 EF	7 Tam
Todos Ramos		med	6,6	31,3	3,4	31,0	6,4	30,0	4,0	30,7	1,3	31,1	1,6	30,6	1,4	30,2
		des	1,5	2,9	1,4	2,9	1,5	3,1	1,4	3,5	0,9	3,6	0,7	3,0	0,9	2,8
Todos P-usos	0	med	11,5	53,2	5,8	53,0	14,2	54,8	8,2	52,8	3,5	51,0	3,8	55,1	3,2	51,3
		des	2,1	4,8	1,6	5,0	2,5	5,9	2,2	5,1	1,1	5,7	1,1	5,2	1,1	5,0
	1	med	11,8	56,1	6,2	54,4	13,7	55,7	7,7	53,6	3,6	52,1	3,9	56,5	3,2	53,2
		des	2,3	5,2	1,9	5,5	2,3	4,3	2,0	4,6	1,2	4,4	1,0	5,2	1,2	4,8
	2	med	11,8	55,0	5,6	53,9	13,6	55,7	8,0	53,3	3,4	52,4	3,9	56,3	3,2	52,7
		des	2,3	5,2	2,2	5,2	2,6	5,0	2,1	4,5	1,0	5,3	1,0	4,9	1,2	5,3
Todos Usos	0	med	11,5	55,2	6,6	54,5	14,4	56,5	8,2	54,9	3,5	53,1	3,8	57,1	3,3	54,0
		des	2,4	5,8	1,8	5,0	2,5	5,0	2,2	5,3	1,1	4,9	1,1	4,9	1,1	5,1
	1	med	11,7	56,3	6,4	55,4	14,6	57,9	8,7	55,6	3,6	54,2	4,0	57,6	3,4	55,3
		des	2,4	4,8	2,3	5,7	2,3	5,5	2,2	5,3	1,2	5,4	1,1	5,1	1,1	5,3
	2	med	11,6	55,5	6,5	55,5	14,7	58,5	8,1	54,7	3,5	54,0	4,0	57,3	3,2	54,1
		des	1,9	4,1	1,8	6,1	2,6	5,4	2,3	5,0	1,2	5,5	1,0	5,6	1,0	4,9
Todos Potenciais Usos	0	med	17,4	74,2	7,7	69,9	18,0	73,0	10,7	70,3	4,7	70,3	4,0	74,0	3,6	71,4
		des	3,0	6,6	2,0	5,4	3,1	5,8	2,6	5,9	1,3	5,5	1,2	5,7	1,1	5,6
	1	med	17,3	78,4	8,3	75,0	18,3	77,7	11,4	76,3	4,8	75,2	4,0	79,8	3,7	76,9
		des	2,8	6,1	2,2	6,6	2,8	6,4	2,8	6,2	1,5	5,1	1,4	6,5	1,2	6,3
	2	med	18,2	81,0	8,7	78,4	19,4	79,4	12,8	77,1	4,7	76,5	3,9	80,6	3,7	78,4
		des	2,5	6,3	2,0	6,1	2,6	5,4	2,8	6,8	1,2	6,0	1,1	6,2	1,3	6,9
Todos Potenciais Usos/Du	0	med	17,3	78,1	7,9	74,7	18,5	77,5	11,6	76,1	4,9	75,7	4,0	79,1	5,2	75,0
		des	3,0	5,5	2,0	5,9	2,7	5,1	2,4	5,4	1,6	5,8	1,3	6,7	1,2	5,6
	1	med	18,1	84,9	8,6	82,3	19,1	85,2	12,2	83,0	4,8	82,3	4,0	85,6	5,8	82,6
		des	3,0	6,1	2,2	5,3	2,5	5,2	2,6	5,5	1,4	6,0	1,0	5,5	1,1	5,9
	2	med	18,6	87,0	9,0	84,1	19,7	84,9	14,2	83,8	4,8	83,8	4,1	86,5	5,8	83,4
		des	1,7	5,5	2,3	5,6	3,2	6,3	2,5	5,4	1,7	6,2	1,2	5,5	1,1	6,2

Tabela 4. Eficácia (EF) de conjuntos gerados aleatoriamente de diferentes tamanhos (Tam).

Tam		*Defeitos*						
		1	*2*	*3*	*4*	*5*	*6*	*7*
30	med	3,8	3,63	3,43	3,65	0,73	0,46	0,23
	des	1,74	1,7	1,5	1,64	0,72	0,74	0,47
40	med	4,8	4,77	4,9	5,14	0,99	0,48	0,31
	des	2,19	1,93	1,91	2,02	0,96	0,61	0,49
50	med	5,98	6,18	6,31	6,54	1,37	0,58	0,48
	des	2,29	2,06	2,37	2,38	1,12	0,71	0,66
55	med	6,03	6,46	6,67	6,88	1,41	0,65	0,41
	des	2,03	2,25	2,77	2,46	1	0,85	0,62
60	med	7,58	7,22	7,24	7,41	1,81	0,67	0,49
	des	2,43	2,35	2,43	2,31	1,29	0,77	0,76
65	med	7,63	8,02	7,76	8,1	1,58	0,96	0,58
	des	2,33	2,43	2,79	2,81	1,17	0,9	0,76
70	med	8,82	8,53	8,97	8,97	1,81	0,86	0,51
	des	2,76	2,61	2,87	2,63	1,4	0,82	0,69
75	med	8,94	9,08	9,24	9,51	2,02	1,11	0,64
	des	2,35	3,06	2,81	2,67	1,38	1,14	0,73
80	med	9,78	9,61	10,06	9,96	2,19	1,19	0,75
	des	3,11	2,96	3,14	2,76	1,32	0,98	0,73
85	med	10,41	10,54	10,2	10,57	2,18	0,99	0,74
	des	2,89	2,87	3,36	2,79	1,37	0,94	0,85
90	med	11	10,85	11,63	11,15	2,33	1,23	0,74
	des	2,76	3,25	2,66	3,06	1,46	1,09	0,82

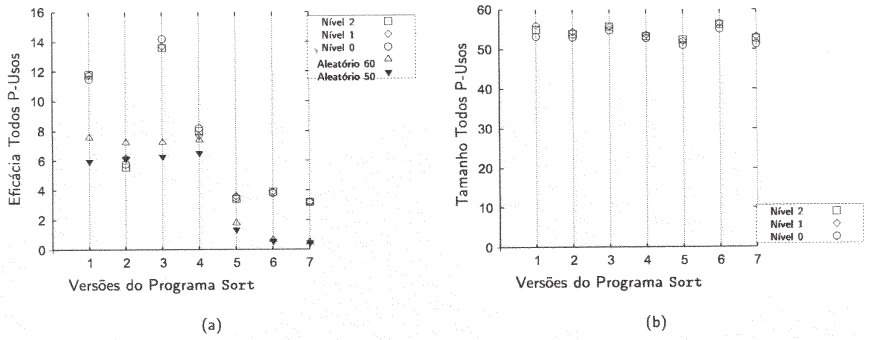

Fig. 2. Eficácia e tamanho dos conjuntos adequados ao critério todos p-usos utilizando os diferentes modelos de dados.

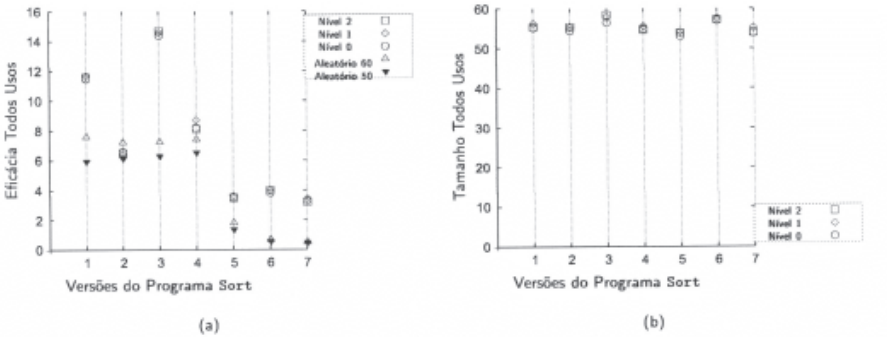

Fig. 3. Eficácia e tamanho dos conjuntos adequados ao critério todos usos utilizando os diferentes modelos de dados.

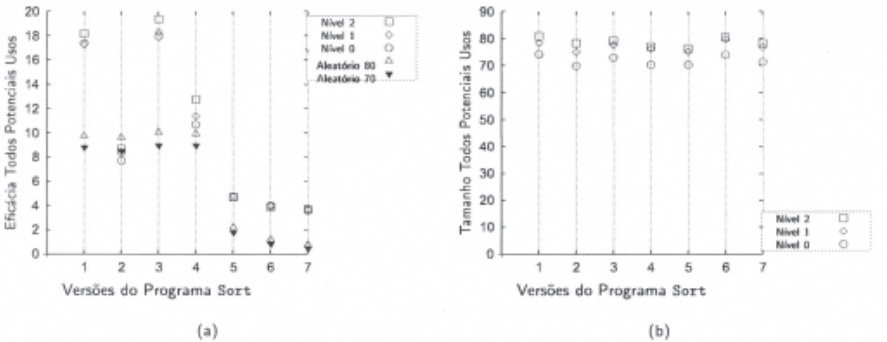

Fig. 4. Eficácia e tamanho dos conjuntos adequados ao critério todos potenciais usos utilizando os diferentes modelos de dados.

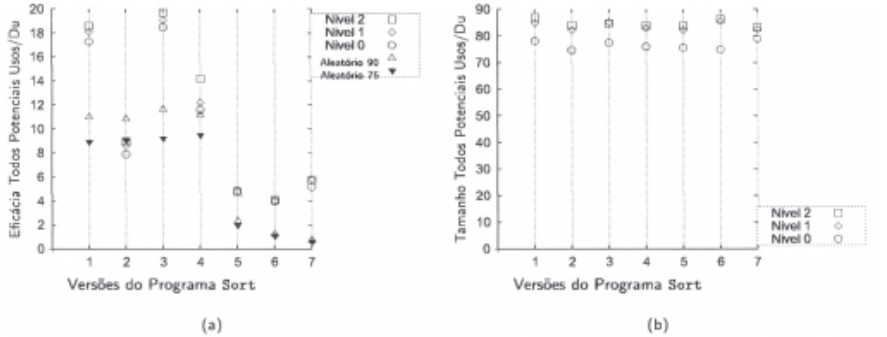

Fig. 5. Eficácia e tamanho dos conjuntos adequados ao critério todos potenciais usos/du utilizando os diferentes modelos de dados.

A partir dos dados coletados, pôde-se observar que o comportamento dos critérios todos p-usos e todos usos, para os programas do experimento, não foi modificado significativamente quando os modelos de dados mais precisos foram utilizados. Note-se nos gráficos das Fig. 2 e 3 que os pontos relativos ao **níveis 0**, **1** e **2** praticamente se sobrepõem. O aumento da eficácia devido aos **níveis 1** e **2** foi sempre menor ou igual a 8% para todos p-usos e a 6% para todos usos. Além disso, para alguns defeitos, o **nível 0** também obteve desempenho ligeiramente melhor (7% para todos p-usos e 4% para todos usos). Com relação ao tamanho dos conjuntos de casos de teste, não houve aumento importante visto que este ficou restringido a 6% para todos p-usos e a 4% para todos usos.

Os critérios da família Potenciais Usos, porém, apresentaram maior sensibilidade aos modelos mais precisos de análise de fluxo de dados. Com relação à eficácia, considerando o critério todos potenciais usos, houve uma versão defeituosa (versão 4) na qual o **nível 2** obteve um desempenho marcadamente melhor que os **níveis 0** e **1** (eficácia 20% maior) e uma outra (versão 2) em que o desempenho foi razoavelmente melhor (eficácia 13% maior); para as demais versões defeituosas, os **níveis 1** e **2** obtiveram melhor eficácia do que o **nível 0**, mas não tão expressivamente melhor. O critério todos potenciais usos/du utilizando o **nível 2** também foi marcadamente melhor para as versões 4 (eficácia 23% maior) e 2 (eficácia 15% maior) enquanto que para as demais versões a eficácia não sofreu modificações marcantes.

O tamanho dos conjuntos adequados aos critérios Potenciais Usos aumentaram consistentemente quando foram utilizados modelos mais precisos como pode ser observado nos gráficos das Fig. 4 (b) e 5 (b). Do **nível 0** ao **nível 2**, o aumento máximo do tamanho dos conjuntos adequados foi de 12% para os dois critérios Potenciais Usos.

Os resultados dos conjuntos de casos de teste gerados aleatoriamente dão uma dimensão do efeito do aumento do número de casos de teste na eficácia. É interessante observar que nem sempre um conjunto de casos de teste maior implica um aumento marcante da eficácia. Isto pode ser observado nas versões 5, 6 e 7. Por outro lado, para algumas versões, um número maior de casos de teste tem um impacto direto na eficácia. Por exemplo, para a versão 2, o impacto na eficácia provocado pelo aumento no número de casos de teste é equivalente ou superior ao provocado pela introdução de modelos de dados mais precisos para os critérios Potenciais Usos. Além

disso, para essa versão, o teste aleatório é mais eficaz que o teste de ramos e de fluxo de dados. Maiores detalhes da versão 2 são discutidos a seguir.

Os resultados apresentados foram obtidos considerando-se algumas aproximações da realidade; por isso, eles devem ser analisados tendo em vista as limitações dessas aproximações. Uma delas é o fato de que se trata de um experimento baseado em um único programa, embora várias versões diferentes tenham sido utilizadas. Todavia, o programa `Sort` é complexo e utiliza extensivamente ponteiros, de forma que o que foi observado nele pode também ocorrer em outros programas de características semelhantes.

Outra limitação é que os defeitos foram semeados artificialmente e são únicos em cada versão. Na prática, observa-se que os programas em geral possuem vários defeitos. Deve-se ressaltar, entretanto, que o propósito do estudo de caso é verificar o desempenho dos diferentes modelos de dados na deteção de defeitos relacionados com ponteiros e campos de registros. Nesse sentido, o isolamento dos defeitos é interessante, pois permite rastrear o relacionamento entre o tipo de defeito e a eficácia do modelo de dados utilizado.

Os resultados devem ainda ser analisados levando em consideração as características da massa de teste e dos conjuntos adequados. A massa foi determinada aleatoriamente; porém, é razoável supor que em muitas situações os dados de entrada serão gerados por testadores, o que deve levar a dados de entrada com características distintas (e.g., concentrados em alguns pontos do domínio de entrada). Os conjuntos selecionados, por sua vez, são *aproximadamente* adequados, pois as associações não-executáveis não foram determinadas.

Dessa maneira, as medidas de eficácia e tamanho dos conjuntos são subestimadas.

A última questão a ser considerada é o fato de os defeitos serem *fáceis* de detectar usando critérios de fluxo de dados. Essa limitação é superada utilizando o número de casos de teste reveladores de defeito como medida de eficácia. Essa medida visa dar uma indicação da habilidade de um modelo de dados em selecionar associações com maior probabilidade de detectar defeitos relativos ao uso incorreto de ponteiros e de campos de registros.

Portanto, é importante observar que os resultados do estudo de caso são influenciados por variáveis como o tipo do defeito, o domínio de aplicação e a estrutura do programa, bem como a arquitetura do sistema no qual ele está inserido.

As versões defeituosas do `Sort` foram inspecionadas para determinar quais as razões do comportamento observado nos resultados. Para os critérios todos p-usos e todos usos, foi verificado que a menor sensibilidade à análise de fluxo de dados mais precisa é devida aos fluxos de controle e de dados particulares ao programa `Sort`. Três situações foram observadas:

- Algumas associações adicionais requeridas pelos **níveis 1** e **2** foram *incluídas* (de acordo com a definição de Marré & Bertolino (1996)) pelas associações de **nível 0**, o que significa dizer que qualquer caminho que exercita a associação **nível 0** também exercita uma ou mais associações adicionais **nível 1** ou **nível 2**. O caso típico envolve associações cujas variáveis compartilham um relacionamento baseado na linguagem de programação (e.g., ponteiro `p` e a variável derreferenciada `*p`). Muito freqüentemente no programa `Sort`, o caso de teste que satisfaz a associação relativa a `p` também satifaz a associação relativa a `*p`.

- Outras associações de **nível 1** ou **nível 2** foram, por sua vez, *incluídas* pelas associações de **nível 0** devido a *condições de executabilidade*. O que ocorre neste caso é que aqueles caminhos que exercitariam as associações de **níveis 1** e **2**, mas não as de **nível 0**, são não-executáveis. Interessantemente, em geral, as variáveis nessas associações não compartilham relação baseada na linguagem de programação; ao contrário, as associações compartilham padrões de não-executabiliade contidos no código do `Sort` que terminam por determinar a relação de inclusão *dinâmica*.

- Há ainda um grupo final de associações **níveis 1** e **2** que é *quase incluído* pelas associações **nível 0**. Essas associações não são incluídas nem estaticamente nem dinamicamente. Mesmo assim, elas possuem uma grande probabilidade de também serem incluídas, pois as condições que evitariam a inclusão são muito particulares. Por exemplo, uma associação adicional **nível 1** somente não seria incluída por outra associação **nível 0** se o laço que controla ambas as associações fosse executado uma única vez em todos os casos de teste.

Dessa maneira, como poucos novos caminhos além daqueles necessários para satisfazer as associações **nível 0** são exigidos, tanto a eficácia como o tamanho dos conjuntos de casos de teste adequados permaneceram praticamente constantes. Portanto, para as versões defeituosas do `Sort`, o custo dos componentes relacionados com o aumento no número de casos

de teste são irrelevantes. Por outro lado, o custo associado com a análise de adequação (C_4) aumenta consideravelmente uma vez que o número de *adu*s pode aumentar em até 90% do **nível 0** para o **nível 2**.

Entretanto, o número de *adu*s pode ser reduzido visto que muitas das associações de **níveis 1** e **2** relativas às variáveis derreferenciadas são incluídas pelas associações de **nível 0** relativas aos ponteiros. Essas associações podem ser abstraídas em uma única associação de maneira semelhante às *adpu*s dos critérios Potenciais Usos. Uma abordagem mais geral, porém, seria determinar a relação de inclusão entre *adu*s utilizando o algoritmo proposto por Marré & Bertolino (1996).

Com relação aos critérios Potenciais Usos, observou-se que as novas *adpu*s exigidas pelos **níveis 1** e **2** são relevantes visto que elas são novos requisitos de teste que requerem novos casos de teste para serem satisfeitos. Este fato provoca aumento no tamanho dos conjuntos adequados de casos de teste. Como conseqüência, os custos influenciados pela cardinalidade dos conjuntos adequados de teste aumentam na mesma taxa. Além disso, o número maior de *adpu*s aumenta também os custos da análise de adequação.

Diferentemente dos critérios todos p-usos e todos usos, a eficácia dos critérios da família Potenciais Usos para os programas do experimento mostrou-se sensível aos defeitos. Para a maioria deles (seis de sete), as associações devidas aos **níveis 1** e **2** não induziram à seleção de casos de teste adicionais que aumentassem marcadamente a eficácia. Entretanto, para a versão defeituosa 4, o aumento na eficácia foi de aproximadamente 20%, o que pode ser considerado significativo. A eficácia também aumentou na versão defeituosa 2 usando todos potenciais usos/du e o **nível 2**; porém, este aumento pode ser creditado ao aumento no número de casos de teste e não ao modelo de dados mais preciso.

A versão 4 do programa `Sort` foi inspecionada para determinar qual o papel do modelo de dados **nível 2** no aumento da eficácia. Para tanto, a *probabilidade* (referenciada como PRD) de cada *adpu* ser exercitada por um caso de teste *revelador de defeito* foi determinada. Como esperado, diferentes probabilidades foram encontradas para as *adpu*s de **níveis 0**, **1** e **2**. Várias associações de **nível 2** obtiveram valores de PRD altos; entretanto, há uma *adpu* desse nível em particular cuja PRD foi de 94%, enquanto que a *adpu* de **nível 0** que obteve melhor PRD atingiu 88%. É interessante observar, porém, que esta associação de **nível 2** ((31,(27,28), { `*i, **i` }) é exigida exclusivamente devido à análise de fluxo de dados mais precisa imposta pelo **nível 2** visto que ela não contém variável definida que seja escalar ou

agregada. Dessa maneira, os caminhos que exercitam a associação e que possuem grande probabilidade de revelar o defeito foram induzidos pelo modelo de dados **nível 2**.

O desempenho dos critérios Potenciais Usos utilizando **nível 2** para a versão 2 é explicado quando são analisados os dados contidos na Tabela 4 e nas Fig. 4 e 5. Comparando o teste de ramos e de fluxo de dados com o teste aleatório (Tabela 4), observa-se que ambos obtiveram melhor desempenho do que o teste aleatório para os defeitos inseridos no `Sort`, com exceção do defeito da versão 2. Para esta versão, o aumento no tamanho dos conjuntos de casos de teste provoca uma melhora observável da eficácia que é equivalente ou superior à causada pelos uso de modelos de dados mais precisos nos critérios potenciais usos.

A razão para este comportamento é que as falhas nesta versão são observadas sempre que uma fusão (*merge*) de arquivos ocorre. Acontece que a chance de serem escolhidos dados de entrada que requeiram uma fusão de arquivos é maior utilizando teste aleatório do que utilizando critérios de teste estruturais. Isto porque o teste aleatório pode escolher vários dados de entrada de um mesmo subdomínio da entrada, o que em princípio não deve ocorrer nos critérios estruturais. Portanto, o aumento da eficácia pode não ter sido causado pelo uso do **nível 2**, mas pelo maior número de casos de teste. Já em relação à versão 4, o aumento da eficácia provocado pelo uso do **nível 2** supera aquele causado pelo simples aumento do número de casos de teste.

Conclusões

Neste trabalho foram apresentadas a definição e a implementação de dois modelos de dados mais precisos para apoiar o teste de programas que utilizam ponteiros e campos de registros. Esses dois novos modelos tiveram a sua *eficácia* e *custo* avaliados por meio de um estudo de caso no qual a base de comparação foi um modelo que não considera os fluxos de dados relativos à derreferenciação de ponteiros ou a campos de registros. No experimento realizado, sete versões *defeituosas* do programa Sort do ambiente Unix foram utilizadas. Este programa foi selecionado porque se trata de um programa complexo que utiliza extensivamente ponteiros e registros.

O estudo de caso realizado indica que a eficácia na detecção de defeitos relativos ao uso incorreto de ponteiros e campos de registros depende do

programa e do *defeito*. Para os critérios da família Fluxo de Dados, a estrutura do programa foi o fator determinante para os resultados observados, enquanto que as características dos defeitos também contribuíram para o desempenho dos critérios Potenciais Usos. Portanto, esta última observação indica que existem defeitos que podem ser mais facilmente detectados quando modelos de dados mais precisos são utilizados. Ainda com relação aos critérios Potenciais Usos, o custo adicional é razoável porque o número de associações requeridas não cresce muito.

De acordo com Harrold (2000), o teste de *todos* os fluxos de dados relativos ao uso de ponteiros pode ser muito caro. Daí a necessidade de experimentos que avaliem a *eficácia* e o *custo* do teste de fluxo de dados de programas que utilizam esses recursos. Assim, a implementação dos modelos de análise de fluxo de dados mais precisos na ferramenta POKE-TOOL e o experimento realizado representam os passos iniciais para um melhor entendimento do teste de fluxo de dados de programas com ponteiros e registros. No entanto, passos adicionais - representados por experimentos novos e mais completos - são necessários e devem ser conduzidos.

Referências Bibliográficas

AGRAWAL, H.; ALBERI, J. L.; HORGAN, J. R.; LI, J. J.; LONDON, S.; WONG, W. E.; GOSH, S.; WILDE, N. Mining system tests to aid software maintenance. **IEEE Computer**, v. 31, n. 7, p. 64-73, 1998.

CHAIM, M. L. **POKE-TOOL - uma ferramenta para suporte ao teste estrutural de programas baseado em análise de fluxo de dados**. 1991. 186 f. Dissertação (Mestrado) – Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas.

CHAIM, M. L.; MALDONADO, J. C.; JINO, M.; NAKAGAWA, E. Y. POKE-TOOL — estado atual de uma ferramenta para teste estrutural de software baseado em análise de fluxo de dados. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 13.; SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 12., 1998, Maringá. **Caderno de ferramentas**. Maringá: SBC, 1998. p. 37-45.

CHUSHO, T. Test data selection and quality estimation based on the concept of essential branches for program testing. **IEEE Transactions on Software Engineering**, v. 13, n. 5, p. 509-517, 1987.

DELMARO, M. E.; MALDONADO, J. C.; MATHUR, A. P. Interface mutation: an approach for integration testing. **IEEE Transactions on Software Engineering**, v. 27, n. 3, Mar. 228-247, 2001.

DUNCAN, I.; ROBSON, D. An exploratory study of common coding faults in C programs. **Software Maintenance: Research and Practice**, v. 8, p. 241-256, 1996.

FRANKL, F. G.; WEISS, S. N.; WEYUKER, E. J. ASSET—a system to select and evaluate test. In: IEEE CONFERENCE ON SOFTWARE TOOLS, 1985, New York. **Proceedings**... Los Alamitos: IEEE Computer Society Press, 1985. p. 72-79.

FRANKL, F. G.; WEYUKER, E. J. An applicable family of data flow testing criteria. **IEEE Transactions on Software Engineering**, v. 14, n. 10, p. 1483-1495, 1988.

GHEZZI, C.; JAZAYERY, M. **Programming languages concepts**. 2nd ed. New York: John Wiley, 1987.

HARROLD, M. J. Testing: a roadmap. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 22., 2000, Limerick. **Proceedings of the conference on the future of software engineering**. New York: ACM Press, 2000. p. 61-72.

HERMAN, P. M. Data flow approach to program testing. **Australian Computer Journal**, v. 8, n. 3, p. 92-96, 1976.

HORGAN, J. R.; LONDON, S. Data flow coverage and the C language. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING ANALYSIS, 1991, Victoria, Canada. **Proceedings of the symposium on testing, analysis, and verification**. New York: ACM Press, 1991. p. 87-97.

LASKI, J.; KOREL, B. A data flow oriented program testing strategy. **IEEE Transactions on Software Engineering**, v. 9, n. 3, p. 347-354, 1983.

MALDONADO, J. C. **Critérios potenciais usos**: uma contribuição ao teste estrutural de software. 1991. 261 f. Tese (Doutorado) – Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas.

MALDONADO, J. C.; CHAIM, M. L.; JINO, M. Bridging the gap in the presence of infeasible paths: potential uses testing criteria. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SOCIETY, 12., 1992, Santiago. **Proceedings**... [Santiago: s.n., 1992a]. p. 323-340.

MALDONADO, J. C.; VERGÍLIO, S. R.; CHAIM, M.L.; JINO, M. Critérios potenciais usos: análise da aplicação de um benchmark. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWAREM 6., 1992, Gramado. **Anais**... Porto Alegre: UFRGS - Instituto de Informática, 1992b. p. 357-371.

MARRÉ, M.; BERTOLINO, A. Unconstrained duals and their use in achieving all-uses coverage. **ACM SIGSOFT Engineering Notes**, v. 21, n.3, p.147-157, May 1996.

MARX, D. I. S.; FRANKL, F. G. Path-sensitive alias analysis for data flow testing. **Software Testing, Verification and Reliability,** v. 9, p. 51-73, 1999.

MARX, D. I. S.; FRANKL, P. G. The path-wise approach to data flow testing with pointer variables. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS, 1996, San Diego. **Proceedings**... New York: ACM Press, 1996. p. 135-146.

NTAFOS, S. A data flow oriented program testing strategy. **IEEE Transactions on Software Engineering**, v. 10, n. 6, p. 795-803, 1984.

OSTRAND, T. J.; WEYUKER, E.J. Data flow-based test adequacy analysis for languages with pointers. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING ANALYSIS, 1991, Victoria, Canada. **Proceedings of the symposium on testing, analysis, and verification**. New York: ACM Press, 1991. p. 87-97.

PANDE, H.; LANDI, W.; RYDER, B. Interprocedural def-use associations for C systems with single level pointers. **IEEE Transactions on Software Engineering**, v. 20, n, 5, p. 789-810, 1994.

RAPPS, S.; WEYUKER, E. J. Selecting software test data using data flow information. **IEEE Transactions on Software Engineering**, v. 14, n. 4, p. 367-375, 1985.

SILVA, J. B. **PROTESTE+**: ambiente de validação automática da qualidade de software através de técnicas de teste e de métricas de complexidade. 1995. Dissertação (Mestrado) – CPGCC, Universidade Federal do Rio Grande do Sul, Porto Alegre.

URAL, H.; YANG, B. A structural test selection criterion. **Information Processing Letters**, v. 57-163, 1988.

VERGILIO, S. R. **Critérios de teste de software restritos**: uma contribuição para gerar dados de teste mais eficazes. 1997. 133 f. Tese (Doutorado) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas.

VILELA, P.; MALDONADO, J. C.; JINO, M. Data flow based testing of programs with pointers: a strategy based on potential uses. In: INTERNATIONAL SOFTWARE QUALITY WEEK – QW'97, 10., 1997, San Francisco. **Proceedings**... San Francisco: Software Research, 1997.

WONG, W. E. **On mutation and data flow**. 1993. Thesis (Doctor of Philosophy) – Purdue University, West Lafayette.


Embrapa
Informática Agropecuária